




# Pipeline

 Author	 Lee Edith.D
 Last Update	@March 22, 2022

[Introduction to Pipeling](#)

[Processor Performance](#)

[Pipeline](#)

[Overall view of instruction execution](#)

[Pipeline Speedup](#)

[How Pipeline is Implemented](#)

[Pipeline Hazards](#)

[Structure Hazards](#)

[Data Hazards](#)

[Control Hazards](#)

[Branch Prediction](#)

[Alleviate Branch Hazards](#)

[Exceptions](#)

[Handling Multicycle Operations](#)

## Introduction to Pipeling

### Processor Performance

- Decrease critical path
  - pipeline
  - hardware fabrication
    - advance technologies , 5nm —

### Pipeline

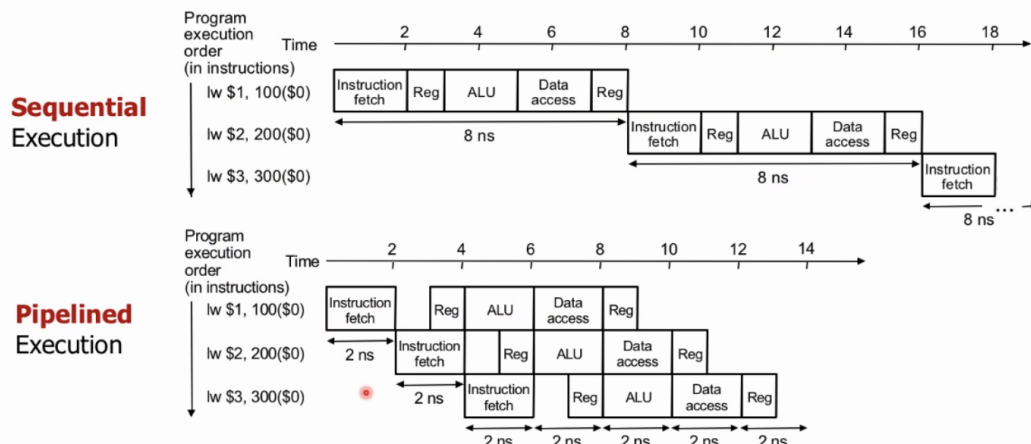
- Allow parallel sub-tasks to be executed simultaneously
- **Don't help latency** ( response time)of a single task
- Help **throughout** of entire work

- **Potential speed up = # of pipeline stages (balance)**
- 5-stage pipeline in MIPS ISA:
  - instruction fetch (IF)
  - register file access ( ID )
  - ALU ( EX )
  - data access ( MEM )
  - register write ( WB )
- all instructions go through 5 stages

## Overall view of instruction execution

Improve performance by increasing instruction **throughput**

Instruction Fetch	Register File Access (Read)	ALU Operation	Data Access	Register Access (Write)
2ns	1ns	2ns	2ns	1ns



## Pipeline Speedup

$$T_{pipeline} = \frac{T_{sequence}}{N_{stages}}$$

- Unbalance → speedup is less

- Latency may even increase
- Throughput will increase

## How Pipeline is Implemented

- 4 pipeline registers:
  - IF/ID
  - ID/EX
  - EX/MEM
  - MEM/WB
- IF
  - Instruction fetch and PC increment
- ID
  - Instruction decode and operand fetch from register file and immediate
- EX
  - RegDst
  - ALUOp [1:0]
  - ALUSrc
- MEM
  - Branch → be careful!
  - MemRead
  - MemWrite
- WB
  - MemToReg
  - RegWrite
- Control signals will be forwarded stage by stage

- Pipeline Control: Extend pipeline registers to include control signals. Control signals are generated during ID stage and passed to the subsequent stages through pipeline registers just like data

## Pipeline Hazards

### Structure Hazards

| Structure hazard is a conflict over the use of a resource at the same time

- Suppose the MIPS CPU with a single memory
  - Load/Store requires data access in MEM stage
  - IF requires instruction access from the same memory
    - IF would have to stall for that cycle
    - bubble
- solution
  - duplication

### Data Hazards

| Data is not ready for the subsequent dependent instruction

- Data dependency
  - e.g. instruction memory and data memory
- solution:
  - stall ( bubble )
    - wait the former instruction to be finished
    - hurt performance
  - Forwarding

- Use the data forward ( after EX for example), do not need to wait until write back stage. No need to grab the data finished after the pipeline, they can be fetched earlier.

- mux used to select the correct one

```
add $s0 , $t1 , $t0
```

```
sub $t2, $s0 , $t3
```

- Can't always avoid stalls by forwarding:

```
lw $s0 , 8( $t1 )
```

```
sub $t2, $s0 , $t3
```

- Stall pipeline if data hazard is detected during ID stage by **data hazard detection logic**
- Bubble can be eliminated by compilers
  - reorder code

## Control Hazards

fetch next instruction depends on current one

when executing a branch instruction, what are the next 2 instructions doing after BRANCH?

- solution: flush the pipeline
- reduce control hazard
  - resolve target in ID stage
  - delayed branch (requires compiler to schedule)
    - compiler filled the delay slot with some instructions that must to be executed

## Branch Prediction

- in some real chip, stall penalty becomes unacceptable because branch instructions are used so frequently
- Solution: Branch Prediction

- implemented in hardware
- do prediction in ID
  - true → branch destination
  - false → PC+4
  - if the prediction is not correct → flush the pipeline

## Alleviate Branch Hazards

- IF: flush → after flush, the instruction will be **nop**
  - op field is replaced by **nop**
- squash

## Exceptions

| the normal order of instruction is changed

- may force the CPU to abort the instructions in the pipeline before they complete
- some others use exceptions
  - interrupt
  - fault
- stop and restart
  - force a trap instruction into the pipeline on the next IF
  - Until the trap is taken, turn off all writes for the faulting instructions and for all instructions that follow in the pipeline
  -

## Handling Multicycle Operations

- float **mul** and **add** take 1+ cycles
  - **mul** takes 7

- **add** takes 4
- **div** takes 24 without pipeline
- Data hazards
- Structural hazards
- Precise Exceptions
  - out-of-order
- Scalar Pipeline
- Superpipeline
  - deeper pipeline
  - achieve higher clock rates
- Superscalar
  - issue multiple instructions to the pipeline at the same example
  - need to duplicate function units/ports
-